Journal of Structural Biology 165 (2009) 19–26

Contents lists available at ScienceDirect

Journal of Structural Biology

journal homepage: www.elsevier.com/locate/yjsbi

Exploiting desktop supercomputing for three-dimensional electron microscopy reconstructions using ART with blobs

J.R. Bilbao-Castro [a,*], R. Marabini [b], C.O.S. Sorzano [c], I. García [a], J.M. Carazo [d], J.J. Fernández [a]

[a] Dept. Arquitectura de Computadores y Electrónica, Universidad de Almería, 04120 Almería, Spain
[b] Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain
[c] Dept. Ingeniería de Sistemas de Información y Comunicaciones, Univ. San Pablo-CEU, Campus Urb. Montepríncipe s/n, 28668 Boadilla del Monte, Madrid, Spain
[d] Centro Nacional de Biotecnología – CSIC, Campus Universidad Autónoma. 28049 Madrid, Spain

A R T I C L E I N F O

Article history:
Received 1 September 2008
Received in revised form 24 September 2008
Accepted 25 September 2008
Available online 8 October 2008

Keywords:
High performance computing
Parallel computing
Iterative reconstruction algorithms
3D Reconstruction
Multi-threading
Multi-core
Three-dimensional electron microscopy
Single particle electron microscopy
ART

A B S T R A C T

Three-dimensional electron microscopy allows direct visualization of biological macromolecules close to their native state. The high impact of this technique in the structural biology field is highly correlated with the development of new image processing algorithms. In order to achieve subnanometer resolution, the size and number of images involved in a three-dimensional reconstruction increase and so do computer requirements. New chips integrating multiple processors are hitting the market at a reduced cost. This high-integration, low-cost trend has just begun and is expected to bring real supercomputers to our laboratory desktops in the coming years. This paper proposes a parallel implementation of a computation-intensive algorithm for three-dimensional reconstruction, ART, that takes advantage of the computational power in modern multicore platforms. ART is a sophisticated iterative reconstruction algorithm that has turned out to be well suited for the conditions found in three-dimensional electron microscopy. In view of the performance obtained in this work, these modern platforms are expected to play an important role to face the future challenges in three-dimensional electron microscopy.

1. Introduction

Three-dimensional electron microscopy (3D-EM) allows structure determination of macromolecular assemblies at subnanometer resolution and, recently, up to near-atomic level (Zhou, 2008). High resolution structural studies demand huge computational costs that derive from the size and number of the images involved as well as the computational complexity of the algorithms. One of the most demanding stages in 3D-EM is 3D reconstruction. Parallel and distributed computing has been traditionally used to cope with those requirements (e.g. Bilbao-Castro et al., 2006; Yang et al., 2007; Fernández, 2008).

Transmission electron microscopy images of thin biological specimens represent two-dimensional projections of the 3D macromolecular structure. From the information contained in a set of EM images a useful estimate of the 3D structure under study can be determined (Fernández et al., 2006; Frank, 2006). The 3D reconstruction problem can then be defined by the following statement: Given a collection of projection images g, determine the 3D structure f that produced the images g. This problem has to be solved under the conditions that the image data, as well as the information about the geometry of data collection that relates g to f, are imperfect; in particular, both the density information in the images and the information regarding the projection direction are corrupted by substantial noise and the contrast transfer function of the electron microscope.

The different three-dimensional reconstruction methods used in 3D-EM are typically classified as transform and series expansion methods. The essence of transform methods is to find a mathematical procedure that describes the recovery of f from its ideal data, and then implement this procedure making use of the actual data. The well-known algorithm called weighted back-projection (WBP) belongs to this family (Radermacher, 1992). Series expansion methods are basically different from a transform method, since no attempt is made to find a mathematical expression for the solution of the original problem. In this case f is approximated by a discretized version that can be expressed as a linear combination of some fixed basis functions (e.g. voxels), and the problem is then modelled as a large system of linear equations to be solved by iterative methods (Herman, 1980, 1998). Algorithms such as ART, CAV or SIRT (Gilbert, 1972; Herman, 1998; Censor et al., 2001) belong to this family. In general, series expansion methods are more robust under noisy situations, but demand more computational

* Corresponding author.
E-mail address: jrbcast@ace.ual.es (J.R. Bilbao-Castro).

resources than the transform methods. This work focuses on block-ART (from now ART) because of its relatively fast convergence rate to the final solution and the reported robust behaviour under noisy conditions (Marabini et al., 1997, 1998; Sorzano et al., 2001; Fernández et al., 2002). ART has been successfully used in numerous experimental studies by 3D-EM (e.g. Llorca et al., 1999).

There exist two main parallelization strategies (Mattson et al., 2004; Fernández, 2008). One of them uses message-passing libraries like MPI (Gropp et al., 1994) or PVM (Geist et al., 1994) to communicate and coordinate processes running on (generally) different interconnected machines (multicomputers) (Hennessy and Patterson, 2007). Its main drawback is that it is difficult to program. An additional drawback of multicomputers is the high communication latencies due to the interconnection networks, depending on the particular architecture of the system. On the other hand, the cost of such systems is usually low as broadly available technologies are used. Better specialized networks could be used but at expenses of increased costs. Parallel implementations of some iterative reconstruction algorithms using message-passing on multicomputers have been successfully used in the field (Fernández et al., 2004; Bilbao-Castro et al., 2006; Yang et al., 2007). This parallelization strategy is well suited to algorithms where communications/synchronizations between different processors do not happen frequently in comparison with processing time. Well suited algorithms are SIRT, CAV and block versions of them (SART, BiCAV, etc.) as they are inherently parallel because multiple images can be concurrently processed (Bilbao-Castro et al., 2006). On the other hand, there exist other iterative algorithms like ART, which are inherently sequential (images are processed one after the other) and parallelism is only possible at a very low level (sub-image level). Such algorithms would not totally benefit from this parallelization strategy because of the substantial communication/synchronization penalties, obtaining low speed-ups.

The other main parallelization strategy consists of using shared-memory machines, where multiple processors share the same memory (multiprocessors) (Hennessy and Patterson, 2007). Thus, parallel processes can communicate with each other through reading/writing from/to memory, which is much faster than using external networks. Such machines have been historically quite expensive due to their limited market and high development costs. Nevertheless, and due to the physical limits being reached on single processor development, general-purpose processor manufacturers, such as Intel, AMD and others, are now encapsulating multiple processors (usually known as cores) within a single chip (Geer, 2005). This has translated rapidly into low-cost, highly efficient, shared-memory desktop machines. This parallelization strategy is well suited for all reconstruction algorithms but is limited by the available number of processors (generally limited by technical reasons). Thus, the main advantages of multi-core machines are its low communication latencies and cost, allowing the development of parallel versions of algorithms, like ART, which cannot benefit from multicomputer strategies.

The shared-memory programming paradigm has been used in this work by means of multi-threading. A thread is a part of a process that is run in parallel along with other parts of the code. Multi-threading can make full use of the new multi-core platforms (Fernández, 2008; Herlihy and Shavit, 2008; Mattson et al., 2004) and any shared-memory computing platform in general. The multi-threaded ART (mt-ART) implementation presented here uses threads to perform calculations in parallel. mt-ART is available at http://xmipp.cnb.csic.es as part of the open source package Xmipp (Marabini et al., 1996; Sorzano et al., 2004).

2. Algorithm and implementation

This section describes the ART algorithm and expounds some basic aspects involved in the parallel implementation.

2.1. The ART algorithm

ART belongs to the series expansion methods. In these methods it is assumed that the solution f may be approximated by the expression:

$$f(\mathbf{r}) \simeq \sum_{j=1}^{J} x_j b_j(\mathbf{r} - \mathbf{r}_j) \tag{1}$$

where $b_j(\mathbf{r} - \mathbf{r}_j)$ is the basis element (e.g. voxels) centered at position $\mathbf{r}_j$. The task then becomes that of estimating the coefficients of the expansion series, that is, the J-dimensional vector $\mathbf{x}$ whose jth component is x_j (see Herman (1980) for details).

In 3D-EM the data collection method is linear so the ith measured image $\mathbf{y}_i$ $(1 \leqslant i \leqslant I)$ can be approximated by:

$$\mathbf{y}_i \simeq \sum_{j=1}^{J} \mathbf{l}_{i,j} x_j \tag{2}$$

where $\mathbf{l}_{i,j}$ is what the ith measurement would be if the structure consisted of only the jth basis function. Our understanding of the data collection procedure usually allows us to calculate (or, at least, to estimate) the $\mathbf{l}_{i,j}$.

The algorithm produces a sequence of J-dimensional vectors $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots$. Typically $\mathbf{x}^{(0)}$ is chosen to be the vector of all zeros, and the process stops after cycling all the data for some integer number of times. In the step going from $\mathbf{x}^{(k)}$ to $\mathbf{x}^{(k+1)}$ we pick the next equality from Eq. (2) to be considered; we denote the index associated with that equality by i_k. Then

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda \frac{\mathbf{y}_{i_k} - \mathbf{l}_{i_k} \cdot \mathbf{x}^{(k)}}{\mathbf{l}_{i_k} \cdot \mathbf{l}_{i_k}} \mathbf{l}_{i_k} \tag{3}$$

where $\mathbf{y}_{i_k}$ is the experimental image considered.

Note that the algorithm, as described above, does not depend on the choice of the basis functions. For this work we have chosen the so-called blob basis functions following the recommendations of Matej and Lewitt (Lewitt, 1990, 1992; Matej and Lewitt, 1995, 1996). A comment to be made is that using blobs as basis functions is efficacious in noisy situations (Matej et al., 1994; Marabini et al., 1997, 1998; Sorzano et al., 2001; Fernández et al., 2002).

2.2. Parallelization of the reconstruction algorithm

In ART the experimental images $\mathbf{y}_{i_k}$ are processed sequentially producing a sequence of progressively refined estimates of the volume ($\mathbf{x}$). Therefore, a parallel approach based on domain decomposition where multiple images are concurrently processed is discarded. A finer grain, lower level, parallel approach is thus necessary where the volume is divided in subdomains (typically as many as available processing units, at least) that are processed in parallel. Our parallel approach for mt-ART, devised for shared-memory platforms, is based on decomposition of the J-dimensional vector $\mathbf{x}$ into T subsets, or slabs of slices, that are processed in parallel by different threads. In the processing of each image $\mathbf{y}_{i_k}$ (see Eq. (3)), there are two barriers to synchronize the threads: one at the beginning and the other at the end of the processing of the image. Furthermore, there is another synchronization operation (via mutual-exclusion) for the computation of the effective projection of the current model $\mathbf{l}_{i_k} \cdot \mathbf{x}^{(k)}$, owing to the data dependency derived from the fact that different basis functions from different slabs are projected to the same point in the projection space. Only when this projection is computed, the backprojection of the error $\mathbf{y}_{i_k} - \mathbf{l}_{i_k} \cdot \mathbf{x}^{(k)}$ can be performed.

Parallel reconstruction based on slab decomposition is not new (Fernández, 2008). There have already been different message-passing strategies for iterative reconstruction methods in 3D-EM also based on decomposition of the volume into independent slabs that are reconstructed in parallel (e.g. SIRT or Conjugate Gradient in Spider (Yang et al., 2007)). Similar strategies have also been used for non-iterative methods, such as WBP in Spider (Yang et al., 2007) or inverse Fourier methods in Bsoft (Heymann and Belnap, 2007) or in AUTO3DEM (Marinescu and Ji, 2003; Yan et al., 2007). This decomposition into slabs, or even into individual slices, are normally used for parallel reconstruction in electron tomography (Perkins et al., 1997; Fernández et al., 2002, 2004, 2008).

Uniform distribution of the workload across different threads is essential to obtain good parallel performance as well as good scalability. Our implementation will be better understood if we note here that: (i) the vector $\mathbf{x}$ represents a volume and it is stored in the computer as a 3D array (this data structure makes the computation of $\mathbf{l}_{i_k}$ easier) and (ii) the shape of the reconstructed volume may be either a sphere or a cube (see Fig. 1).

For cubic shaped reconstruction, load balancing can be obtained by assigning the same number of slices to each thread. Nevertheless, such a load-balancing schema is not efficient if the volume to reconstruct is spherical (see Fig. 1(a)). Such inefficiency comes from the fact that the slice size (number of blobs in the slice) decreases with its distance to the center of the sphere (see Fig. 1(b)) and so does the time needed to process them. To overcome this problem, an on-demand assignment of slices to threads could be implemented. This way, a thread which is processing a low-loaded slice would request a new slice whereas other threads are still working on other, more populated slices. This schema would limit idle threads but would involve an extra overload which could, potentially, affect scalability. An easier solution has been adopted independently of the volume to reconstruct (cubic or spherical): let T be the number of threads and S be the number of slices then, the slice s will be assigned to the thread $t = s \bmod T$ (see Fig. 1(b)).

2.3. Parallelization of blobs-to-voxels conversion

Working with blobs has many advantages in terms of robustness against noise but is not convenient for data visualization since most programs work with voxels. Therefore, a conversion step is desirable after completion of the reconstruction process. This step is rather time-consuming and might even turn out to be the most demanding step when reconstructing large volumes from a small set of images, as in tomography (Fernández et al., 2002) or 2D-Crystals (Marabini et al., 2004).

The conversion is made by evaluating Eq. (1) at the center $\mathbf{r}_j$ of each voxel (which are placed in different positions from those used by the blobs). This evaluation is performed by adding the contribution of each blob to each voxel. In the sequential case the implementation of this algorithm is straightforward but for the parallel case special care must be taken to avoid voxel values being modified at the same time by two different threads (see Fig. 2 for details).

As in the parallelization of the reconstruction algorithm, all the blobs belonging to a slice are assigned to the same thread. Nevertheless, this time an on-demand policy is used where a thread will request a new slice as soon as it finishes the previously assigned one and becomes idle. A potential conflict may appear when two threads process neighbor slices. To tackle this problem, an explicit mechanism controls that a sufficient separation (in terms of overlapping blobs) exists between slices being processed at a time. Such separation will depend on the size chosen by the user for the blob radius. The program controls the separation issue by using a status vector (see Fig. 3).

3. Experimental evaluation

To test the parallel performance, four phantoms of sizes: $64 \times 64 \times 64$, $128 \times 128 \times 128$, $256 \times 256 \times 256$ and $512 \times 512 \times 512$ were created. Such sizes cover the most common scenarios on electron microscopy studies. For each phantom, a set of 100 projection images were produced. The number of images in typical 3D-EM studies varies from a few thousands to hundreds of thousands but, since the parallelization is made by dividing the volume in subsets, the actual number of images is not relevant for parallel performance measurements. In addition to these tests we have checked that the algorithm performs appropriately with experimental data. One experimental test with an icosahedral virus (see San Martín et al., 2008 for details) is reported in this article. A total of 190,260 projections (3171 experimental projections and 60-fold symmetry) with a size of 275×275 pixels were used.

The experiments were performed in two shared-memory platforms with quite different architectures. We denote them as "Machine8" and "Machine16" and their characteristics are:

- "Machine8": Xeon based Dell Poweredge 1900 workstation, with $2 \times$ Xeon E5320 Quad-Core, counting for a total of 8 processing "cores", and a total amount of 16 GB of RAM memory. The processors were running at 1.86 GHz, with a Level-1 cache

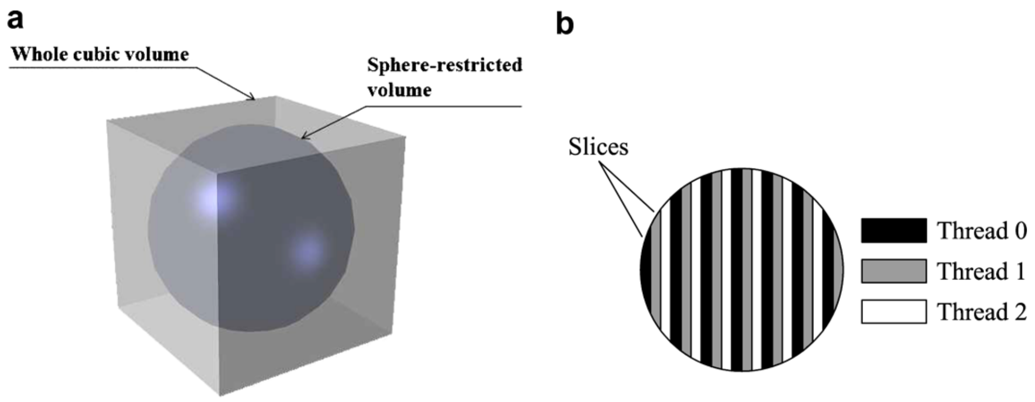

Fig. 1. (a) Composition showing the restriction sphere used to reduce the computing time for a 3D reconstruction. The sphere is centered inside the original cubic volume. (b) Load balancing technique implemented. Threads 0 and 1 will process 7 slices while thread 2 will process 6 slices. Nevertheless, the load is still kept balanced as the slices processed by thread 2 are larger than those assigned to threads 0 and 1.

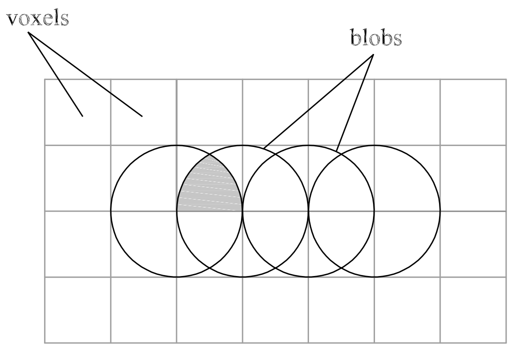

Fig. 2. Blobs (presented here as circles), overlap each other to effectively cover the reconstruction space. Therefore, the value of each voxel (grey cubes) is the result of the contribution of many blobs.

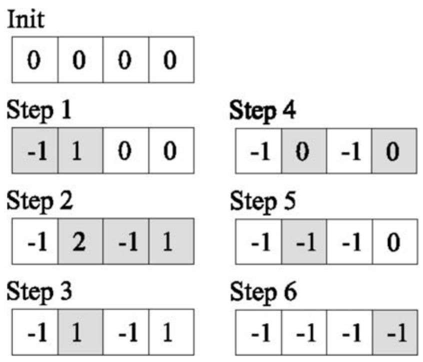

Fig. 3. Mutual-exclusion and load-balancing strategy for blobs-to-voxels conversion. A vector contains the status of each slice. 0 means that such a slice has not been processed yet and that it is eligible for immediate processing. -1 means that the slice is being or has been processed. Positive values mean the slice is locked due to its proximity to one or more slices being processed. When a slice is processed, its entry in the status vector gets a value of -1, and the entries of the immediate neighbors that have not been processed yet are incremented in one unit in order to lock them. When the processing of a slice is finished, the entries of the neighbor, locked slices are decremented in one unit. In this example, we have two threads working to convert four slices. Shaded boxes represent a change in the value. At the first step, one thread starts working on the first slice and locks (to avoid proximity problems and considering a blob radius of 1) the neighbor slice. At the second step, the other thread starts working on the first slice whose value is 0, thus the neighbor slices are locked by incrementing the corresponding entries in the status vector (so the neighbor on the left side gets a value of 2, and that on the right gets a value of 1). At the third step, the first thread finishes with the first slice, and decrements the entry of the neighbor slice in one unit. At the fourth step, the second thread finishes with the slice it was working on, and thus decrements the entries of the neighbor slices. At the fifth step, the first thread starts working on one of the slices not processed yet (i.e. with value 0). In this case, the neighbor slices are not locked since they were already processed (denoted by an entry with value -1). A similar situation happens in step 6.

of 64 KB (per core) and a Level-2 cache of 8 MB (shared, per couple of cores). Cache coherence was maintained through the standard MESI protocol (Hennessy and Patterson, 2007).

- “Machine16”: Itanium2 based SGI Altix 330 machine. It was comprised of 8 interconnected nodes, each containing two processors at 1.5 GHz and 8 GB of RAM. Therefore, the machine had a total of 16 processors and 64 GB of memory. The interconnection is implemented through a high bandwidth proprietary network. Processors had a Level-1 cache of 32 KB, a Level-2 cache of 256 KB and a Level-3 cache of 4 MB. Cache coherence was maintained through the standard MESI protocol (Hennessy and Patterson, 2007). This machine had a scalable shared-memory or distributed-shared-memory architecture, meaning that the physically separate memory (8 nodes with 8 GB RAM memory each) can be addressed as one virtually unique memory system of 64 GB, but the processors have non-uniform memory access (NUMA) as the latency depends on the physical location of the data.

At the time of purchase, the market price of Machine16 was around eight times that of Machine8. Both machines were running the same operating system (openSUSE 10.2). The compiler used was GNU gcc.

For each combination of phantom size and machine, two different types of experiments were carried out. The first type consisted of cubic shaped reconstructions, whereas the second one restricted the reconstructions to a smaller sphere circumscribed by the cubic-volume. For the ART reconstruction, a single loop through the images was used. The parallel experiments were done using a power-of-two number of threads, up to the number of processors available in the machine (8 for Machine8 and 16 for Machine16). For statistical purposes, each experiment was repeated 5 times and the average computing time was measured. For performance assessment, each experiment was also repeated using the original sequential ART implementation running on a single processor.

Prior to the performance evaluation, an analysis of the load balancing scheme proposed for spherical reconstructions was carried out in order to assess its ability to keep a balanced scenario. The number of basis functions processed by each thread, for the different datasets and numbers of threads, was computed. It turned out that the number of blobs processed by each thread was quite similar and the absolute deviations from the average were less than 1%. Therefore, we could conclude that the scheme proposed here managed to balance the workload.

The performance of the parallel algorithms was measured by computing (i) the reconstruction time, (ii) the conversion time and (iii) the speed-up metric. The reconstruction time is the time needed to process all the images and generate the 3D reconstruction expressed as a set of blobs. This time depends on the size of the problem, comprising both the number and size of projections. The conversion time is the time needed to convert the blob volume to a voxel volume, and it only depends on the volume size. Finally, the execution performance and scalability were assessed by means of the speed-up metric, which is denoted as:

$$S_N = T_{seq}/T_N \tag{4}$$

where T_{seq} denotes the execution time for the sequential version of the program and T_N represents the parallel execution time for N threads.

Tables 1 and 2 contain average times and speed-ups obtained by mt-ART for cubic and spherical reconstructions, respectively. The average times and speed-ups for the reconstruction step are also plotted in Figs. 4 and 5, respectively. The time dedicated to the conversion from blobs to voxels has turned out to be significantly lower than the reconstruction from only 100 images. So, it should be expected to be negligible in experimental structural studies involving thousands of images. Therefore, in the following we will focus on the reconstruction time, rather than on the total time, to draw some general, major conclusions.

Tables 1 and 2 clearly point out that, for the same number of threads, the average times for Machine8 are smaller than those obtained for Machine16, for both cubic and spherical reconstruction. Moreover, in general the speed-ups prove to be better for Machine8 than for Machine16. The speed-ups for Machine8 shown in Fig. 5 have turned out to be near-linear with the number of threads/processors used. However, Machine16 exhibits a slightly different behaviour that may be caused by the fact that the access to the memory is not uniform for all of the processors due to the NUMA architecture.

Under ideal conditions, the speed-up for parallel implementations could be equal to the number of processors used. Neverthe-

Table 1
Summary of the results for cubic reconstruction. Average computation times (s) and speed-ups (boldfaced) for the reconstruction and conversion steps are shown as a function of the volume size, number of threads, and the machine used. The speed-up was computed as the ratio between the sequential and the parallel time.

	Machine8						Machine16					
	Reconstruction	Speed-up	Conversion	Speed-up	Total	Speed-up	Reconstruction	Speed-up	Conversion	Speed-up	Total	Speed-up
$64 \times 64 \times 64$												
Sequential	17.10	–	1.06	–	18.17	–	41.72	–	2.55	–	44.23	–
2 threads	8.15	**2.10**	0.63	**1.69**	8.79	**2.07**	21.35	**1.95**	2.76	**0.92**	24.11	**1.83**
4 threads	4.39	**3.89**	0.35	**3.01**	4.75	**3.83**	12.35	**3.38**	1.09	**2.34**	13.44	**3.30**
8 threads	2.80	**6.12**	0.22	**4.90**	3.01	**6.03**	10.89	**3.83**	0.57	**4.48**	11.46	**3.86**
16 threads	–	–	–	–	–	–	9.28	**4.49**	0.67	**3.80**	9.95	**4.44**
$128 \times 128 \times 128$												
Sequential	139.31	–	8.52	–	147.83	–	319.52	–	20.48	–	340.00	–
2 threads	66.59	**2.09**	4.99	**1.71**	71.58	**2.07**	163.51	**1.95**	11.84	**1.73**	175.35	**1.94**
4 threads	34.44	**4.05**	2.69	**3.17**	37.13	**3.98**	86.19	**3.71**	8.95	**2.29**	95.14	**3.57**
8 threads	18.78	**7.42**	1.54	**5.54**	20.32	**7.27**	54.37	**5.88**	4.52	**4.53**	58.89	**5.77**
16 threads	–	–	–	–	–	–	37.12	**8.61**	3.72	**5.50**	40.84	**8.32**
$256 \times 256 \times 256$												
Sequential	1071.72	–	68.24	–	1139.96	–	2495.51	–	163.68	–	2659.19	–
2 threads	511.39	**2.10**	36.93	**1.85**	548.32	**2.10**	1271.26	**1.96**	94.93	**1.72**	1366.20	**1.95**
4 threads	258.76	**4.14**	22.84	**2.99**	281.59	**4.05**	671.36	**3.72**	68.71	**2.38**	740.06	**3.59**
8 threads	143.80	**7.45**	11.35	**6.01**	155.15	**7.35**	442.79	**5.64**	34.27	**4.78**	477.06	**5.57**
16 threads	–	–	–	–	–	–	281.28	**8.87**	20.52	**7.98**	301.80	**8.81**
$512 \times 512 \times 512$												
Sequential	8614.65	–	548.77	–	9163.43	–	19906.51	–	1316.18	–	21222.69	–
2 threads	4101.07	**2.10**	291.64	**1.88**	4392.71	**2.09**	10120.29	**1.97**	755.56	**1.74**	10875.85	**1.95**
4 threads	2068.89	**4.16**	158.43	**3.46**	2227.32	**4.11**	5121.90	**3.89**	551.63	**2.39**	5673.53	**3.74**
8 threads	1103.46	**7.81**	89.10	**6.16**	1192.57	**7.68**	3355.56	**5.93**	254.14	**5.18**	3609.70	**5.88**
16 threads	–	–	–	–	–	–	2201.40	**9.04**	151.81	**8.67**	2353.21	**9.02**

Table 2
Summary of the results for spherical reconstruction. Average computation times (s) and speed-ups (boldfaced) for the reconstruction and conversion steps are shown as a function of the volume size, number of threads, and the machine used. The speed-up was computed as the ratio between the sequential and the parallel time.

	Machine8						Machine16					
	Reconstruction	Speed-up	Conversion	Speed-up	Total	Speed-up	Reconstruction	Speed-up	Conversion	Speed-up	Total	Speed-up
$64 \times 64 \times 64$												
Sequential	9.89	–	0.58	–	10.46	–	25.34	–	1.40	–	26.74	–
2 threads	4.71	**2.10**	0.34	**1.76**	5.05	**1.94**	12.97	**1.95**	0.81	**1.75**	13.78	**1.94**
4 threads	2.70	**3.66**	0.20	**2.95**	2.90	**3.37**	7.07	**3.58**	1.28	**1.11**	8.35	**3.21**
8 threads	1.73	**5.72**	0.14	**4.38**	1.86	**5.24**	6.12	**4.14**	0.40	**3.55**	6.52	**4.11**
16 threads	–	–	–	–	–	–	5.62	**4.51**	0.29	**4.88**	5.91	**4.53**
$128 \times 128 \times 128$												
Sequential	82.48	–	4.61	–	87.09	–	204.24	–	11.28	–	215.51	–
2 threads	40.13	**2.06**	2.63	**1.81**	42.76	**1.94**	103.74	**1.97**	5.98	**1.89**	109.71	**1.97**
4 threads	21.81	**3.78**	1.52	**3.13**	23.33	**3.56**	58.83	**3.47**	6.59	**1.71**	65.42	**3.30**
8 threads	11.68	**7.06**	0.92	**5.17**	12.60	**6.60**	44.30	**4.61**	2.73	**4.13**	47.03	**4.58**
16 threads	–	–	–	–	–	–	23.75	**8.60**	1.93	**5.86**	25.67	**8.40**
$256 \times 256 \times 256$												
Sequential	647.44	–	37.02	–	684.46	–	1633.42	–	90.90	–	1724.31	–
2 threads	314.84	**2.06**	20.45	**1.86**	335.29	**1.95**	832.00	**1.96**	50.36	**1.80**	882.35	**1.95**
4 threads	163.97	**3.95**	12.14	**3.13**	176.11	**3.72**	470.89	**3.47**	55.07	**1.65**	525.96	**3.28**
8 threads	91.71	**7.06**	7.12	**5.33**	98.83	**6.63**	354.79	**4.60**	23.94	**3.79**	378.73	**4.55**
16 threads	–	–	–	–	–	–	194.50	**8.40**	13.19	**6.88**	207.70	**8.30**
$512 \times 512 \times 512$												
Sequential	5260.07	–	302.24	–	5562.30	–	13050.93	–	731.49	–	13782.42	–
2 threads	2552.30	**2.06**	163.55	**1.87**	2715.85	**1.96**	6540.15	**1.97**	395.92	**1.88**	7036.07	**1.96**
4 threads	1281.50	**4.10**	96.04	**3.18**	1377.54	**3.87**	3415.78	**3.82**	357.52	**2.08**	3773.30	**3.65**
8 threads	706.61	**7.44**	57.25	**5.34**	763.86	**6.97**	2535.67	**5.15**	163.00	**4.56**	2698.66	**5.11**
16 threads	–	–	–	–	–	–	1495.56	**8.73**	99.47	**7.46**	1595.03	**8.64**

less, there exist different aspects that make the real scenario not so favorable and limit the theoretical maximum speed-up, as stated by the well-known Amdahl's Law in the computer architecture field. In this case, for example, the initialization of variables and other structures is sequential, meaning that such part of the code will not benefit from parallelization. Also, synchronization points exist in the code that make threads to stay idle from time to time waiting for other operations to be finished. In that sense, mt-ART on Machine8 exhibits near optimal speed-up. A point worth commenting is the slight super-speed-up behaviour for few threads on Machine8, which may derive from a better exploitation of the on-chip cache hierarchy.

On the other hand, the speed-up curves show an improvement with the problem size, regardless of the platform. The higher ratio of computation versus synchronization among threads is the underlying cause. Similarly, cubic reconstructions have also shown

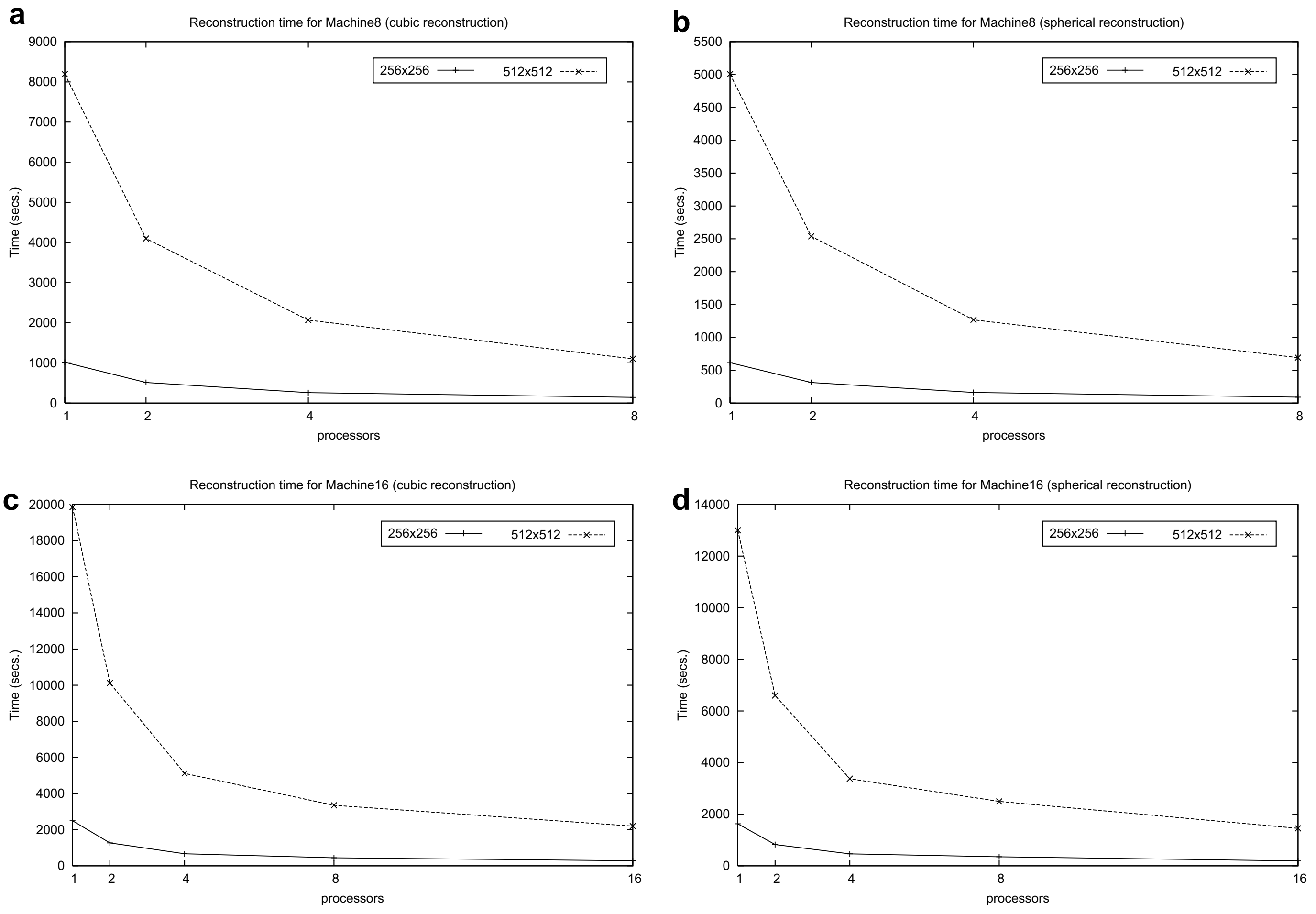

Fig. 4. Curves representing the average time required to perform the reconstruction for: (a) Machine8 and cubic reconstruction, (b) Machine8 and spherical reconstruction, (c) Machine16 and cubic reconstruction and (d) Machine16 and spherical reconstruction.

higher speed-ups than those confined to a sphere, owing to the higher computation–synchronization ratio. So, the cubic case scales better than the spherical one. Nevertheless, in practice, the user will look for the fastest response and that will come from the spherical reconstruction. For the most demanding case, the $512 \times 512 \times 512$ volume, we get a reconstruction time of 1103.46 s for Machine8 and eight processors and 3355.56 s for Machine16 and 8 processors for the cubic case. In contrast, for the spherical case, we get the final result in just 706.61 s for Machine8 and 8 processors and 2535.67 for Machine16 and 8 processors. This represents an improvement factor of 1.6 and 1.3 for Machine8 and Machine16, respectively, when comparing the cubic versus the spherical reconstructions times.

Therefore, in view of the average reconstruction times and the speed-up curves, Machine8 outperforms Machine16 on the execution of mt-ART and, due to its much lower purchase cost, has a significantly better performance-to-cost ratio. Moreover, mt-ART on Machine8 succeeds in achieving a reduction of computation time that is almost linear with the number of processors used.

The parallel implementation of ART has also been tested on experimental data of adenovirus (San Martín et al., 2008). Fig. 6 shows the reconstruction of the icosahedral virus using a total of 190,260 projections (3171 experimental projections, 60-fold symmetry) with a size of 275×275 pixels. With the original, sequential implementation of ART available on Xmipp, the execution time on the best machine (Machine8), for the cubic reconstruction, was 27.87 days. When run using mt-ART with eight threads on the same machine, the execution time was 3.77 days. Therefore, the speed-up obtained for this experimental case was around 7.37. In the case of the spherical reconstruction, the sequential ART execution took 13.14 days, while the same experiment running on 8 processors took 1.92 days, which represents a speed-up of 6.92. These results are consistent with the speed-up figures obtained with phantom data.

4. Conclusions

The work presented here has shown that it is possible to significantly accelerate 3D reconstructions using iterative reconstruction algorithms such as ART even without a high investment on dedicated, specific-purpose high performance hardware. In that sense, this work even demonstrates that a relatively simple desktop workstation using recent technology can yield better performance than eight times more expensive hardware designed for parallel computation.

In a few years to come, it is expected that the number of processors integrated in a single chip will grow exponentially, giving laboratories the possibility to perform high performance computing at a fraction of the present cost. In fact, new consumer hardware is already being equipped with four cores chips at a really low cost. Also, memory and hard disk storage prices drop, leading to more powerful, inexpensive machines.

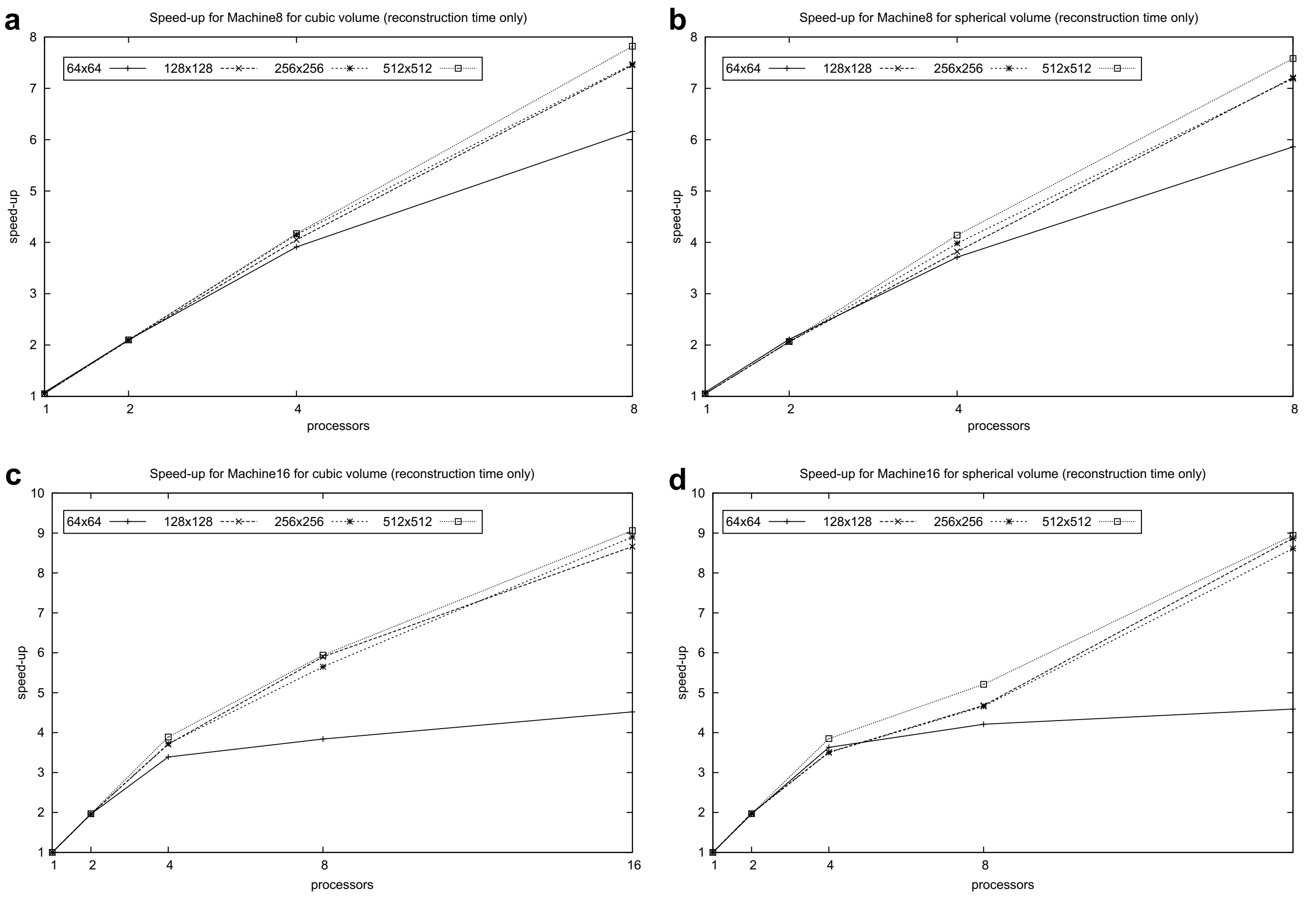

Fig. 5. Curves representing the speed-up obtained for the reconstruction step for: (a) Machine8 and cubic reconstruction, (b) Machine8 and spherical reconstruction, (c) Machine16 and cubic reconstruction and (d) Machine16 and spherical reconstruction.

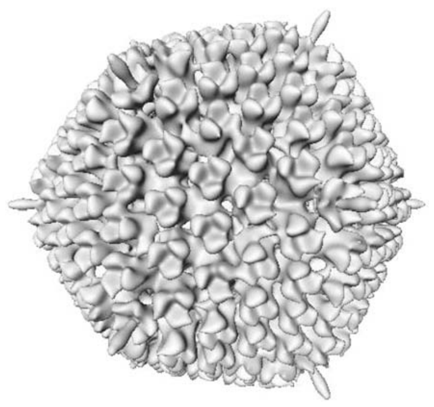

Fig. 6. 3D reconstruction of the adenovirus capsid obtained using mt-ART with eight threads on Machine8, 190,260 projections and one cycle through the data. It was completed on 3.77 days versus 27.87 days for the sequential version of the program.

Therefore, the authors suggest that this kind of "affordable" parallelism should be taken into account for current and future developments in the field of 3D-EM. The authors also feel that high performance computing on these modern platforms is going to play an important role for the future 3D-EM challenges like structural elucidation of large macromolecular assemblies at atomic resolution.

Acknowledgments

The authors wish to thank Dr. Carmen San Martín and collaborators for kindly providing the adenovirus dataset. Work partially funded by grants from the Spanish MEC (TIN2005-00447, TIN2008-01117, BIO2007-67150-C03-01/03), J. Andalucía (P06-TIC-01426), Fondo de Investigación Sanitaria (FIS 04/0683) and EU (LSHG-CT-2004-502828). J.R.B.-C. is a fellow of the Spanish "Juan de la Cierva" postdoctoral contracts program, co-financed by the European Social Fund.

References

Bilbao-Castro, J.R., Carazo, J.M., García, I., Fernández, J.J., 2006. Parallelization of reconstruction algorithms in three-dimensional electron microscopy. Appl. Math. Model. 30, 688–701.

Censor, Y., Gordon, D., Gordon, R., 2001. Component averaging: an efficient iterative parallel algorithm for large and sparse unstructured problems. Parallel Comput. 27, 777–808.

Fernández, J.J., 2008. High performance computing in structural determination by electron cryomicroscopy. J. Struct. Biol. 164, 1–6.

Fernández, J.J., Lawrence, A.F., Roca, J., García, I., Ellisman, M.H., Carazo, J.M., 2002. High performance electron tomography of complex biological specimens. J. Struct. Biol. 138, 6–20.

Fernández, J.J., Carazo, J.M., García, I., 2004. Three-dimensional reconstruction of cellular structures by electron microscope tomography and parallel computing. J. Parallel Distrib. Comput. 64, 285–300.

Fernández, J.J., Sorzano, C.O.S., Marabini, R., Carazo, J.M., 2006. Image processing and 3-D reconstruction in electron microscopy. IEEE Signal Process. Mag. 23 (3), 84–94.

Fernández, J.J., Gordon, D., Gordon, R., 2008. Efficient parallel implementation of iterative reconstruction algorithms for electron tomography. J. Parallel Distrib. Comput. 68, 626–640.

Frank, J., 2006. Three-Dimensional Electron Microscopy of Macromolecular Assemblies. Oxford University Press, Oxford.

Geer, D., 2005. Chip makers turn to multicore processors. Computer 38, 11–13.

Geist, A., Bequelin, A., Dongarra, J., Jiang, W., Mancheck, R., Sunderam, V.S., 1994. PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Network Parallel Computing. MIT Press, Cambridge, MA.

Gilbert, P., 1972. Iterative methods for the 3D reconstruction of an object from projections. J. Theor. Biol. 76, 105–117.

Gropp, W., Lusk, E., Skjellum, A., 1994. Using MPI Portable Parallel Programming with the Message-Passing Interface. MIT Press, Cambridge, MA.

Hennessy, J.L., Patterson, D.A., 2007. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Los Altos, CA.

Herlihy, M., Shavit, N., 2008. The Art of Multiprocessor Programming. Morgan Kaufmann, Los Altos, CA.

Herman, G.T., 1980. Image Reconstruction from Projections: The Fundamentals of Computerized Tomography. Academic Press, New York.

Herman, G.T., 1998. Algebraic reconstruction techniques in medical imaging. In: Leondes, C. (Ed.), Medical Imaging. Systems, Techniques and Applications. Gordon and Breach Science, New York, pp. 1–42.

Heymann, J.B., Belnap, D.M., 2007. Bsoft: image processing and molecular modeling for electron microscopy. J. Struct. Biol. 157, 3–18.

Lewitt, R.M., 1990. Multidimensional digital image representation using generalized Kaiser–Bessel window functions. J. Opt. Soc. Am. A7, 1834–1846.

Lewitt, R.M., 1992. Alternatives to voxels for image representation in iterative reconstruction algorithms. Phys. Med. Biol. 37, 705–716.

Llorca, O., Mccormack, E.A., Hynes, G., Grantham, J., Cordell, J., Carrascosa, J.L., Willison, K.R., Fernández, J.J., Valpuesta, J.M., 1999. Eukaryotic type II chaperonin CCT interacts with actin through specific subunits. Nature 402, 693–696.

Marabini, R., Masegosa, I.M., San Martín, M.C., Marco, S., Fernández, J.J., de la Fraga, L.G., Vaquerizo, C., Carazo, J.M., 1996. Xmipp: an image processing package for electron microscopy. J. Struct. Biol. 116, 237–240.

Marabini, R., Rietzel, E., Schroeder, E., Herman, G.T., Carazo, J.M., 1997. Three-dimensional reconstruction from reduced sets of very noisy images acquired following a single-axis tilt schema. J. Struct. Biol. 120, 363–371.

Marabini, R., Herman, G.T., Carazo, J.M., 1998. 3D reconstruction in electron microscopy using ART with smooth spherically symmetric volume elements (blobs). Ultramicroscopy 72, 53–56.

Marabini, R., Sorzano, C.O.S., Matej, S., Fernández, J.J., Carazo, J.M., Herman, G.T., 2004. 3D reconstruction of 2D crystals in real space. IEEE Trans. Image Process. 13, 549–561.

Marinescu, D.C., Ji, Y., 2003. A computational framework for the 3D structure determination of viruses with unknown symmetry. J. Parallel Distrib. Comput. 63, 738–758.

Matej, S., Lewitt, R.M., 1995. Efficient 3D grids for image reconstruction using spherically symmetric volume elements. IEEE Trans. Nucl. Sci. 42, 1361–1370.

Matej, S., Lewitt, R.M., 1996. Practical considerations for 3D image reconstruction using spherically symmetric volume elements. IEEE Trans. Med. Image 15, 68–78.

Matej, S., Herman, G.T., Narayan, T.K., Furuie, S.S., Lewitt, R.M., Kinahan, P.E., 1994. Evaluation of task-oriented performance of several fully 3D PET reconstruction algorithms. Phys. Med. Biol. 39, 355–367.

Mattson, T.G., Sanders, B.A., Massingill, B.L., 2004. Patterns for Parallel Programming. Addison-Wesley Professional, Reading, MA.

Perkins, G.A., Renken, C.W., Song, J.Y., Frey, T.G., Young, S.J., Lamont, S., Martone, M.E., Lindsey, S., Ellisman, M.H., 1997. Electron tomography of large, multicomponent biological structures. J. Struct. Biol. 120, 219–227.

Radermacher, M., 1992. Weighted back-projection methods. In: Frank, J. (Ed.), Electron Tomography. Plenum Press, New York, pp. 91–115.

San Martín, C., Glasgow, J.N., Borovjagin, A., Beatty, M.S., Kashentseva, E.A., Curiel, D.T., Marabini, R., Dimitriev, I.P., 2008. Localization of the N-terminus of minor coat protein IIIa in the adenovirus capsid. J. Mol. Biol. 383, 923–934. doi:10.1016/j.jmb.2008.08.054.

Sorzano, C.O.S., Marabini, R., Boisset, N., Rietzel, E., Schroder, R., Herman, G.T., Carazo, J.M., 2001. The effect of overabundant projection directions on 3D reconstruction algorithms. J. Struct. Biol. 133, 108–118.

Sorzano, C.O.S., Marabini, R., Velázquez-Muriel, J., Bilbao-Castro, J.R., Scheres, S.H., Carazo, J.M., Pascual-Montano, A., 2004. Xmipp: a new generation of an open-source image processing package for electron microscopy. J. Struct. Biol. 148, 194–204.

Yan, X., Sinkovits, R.S., Baker, T.S., 2007. AUTO3DEM – an automated and high throughput program for image reconstruction of icosahedral particles. J. Struct. Biol. 157, 73–82.

Yang, C., Penczek, P.A., Leith, A., Asturias, F.J., Ng, E.G., Glaeser, R.M., Frank, J., 2007. The parallelization of SPIDER on distributed-memory computers using MPI. J. Struct. Biol. 157, 240–249.

Zhou, Z.H., 2008. Towards atomic resolution structural determination by single-particle cryo-electron microscopy. Curr. Opin. Struct. Biol. 18, 218–228.